Access IRM Data File

via RETDAT

Mon, Feb 17, 2003

The nonvolatile memory file system in IRMs is normally accessed by an 8-character file name and an offset. The ident is therefore quite long and cannot be squeezed into an SSDN for use with the RETDAT protocol. This note describes how a client application can access the contents of a file from an IRM 68K-based front end. The motivation for this came from the desire to access the HELPLOOP data file that contains prompting text for describing the parameters of a local application.

The scheme is based upon searching through the CODES table, which is the file system directory, for a match on the desired file name. Within the 32-byte CODES table entry, there are fields that include the nonvolatile memory base address of the file, the file size, and the file version date. After finding a match, a series of memory accesses can be used to retrieve the file contents. Here are the details of how this is done.

To access the CODES table, listype 75 can be used, which assumes an ident that contains the CODES table entry number. Consider the following SSDN:

```
4B11 0508 0000 0000
```

The CODES table in node0508 is extra large, containing 256 entries. This SSDN would permit reading out the entire CODES table with a request for 8192 bytes. If this is too difficult for the client, the table could be read out in pieces, using the Offset word to indicate the starting entry number for each piece. (The second half of this file could be read out using an offset value of 128 and a length of 4096 bytes.) Once the entire table has been read, one has an array of 32-byte entries. Each entry has the following layout:

Field	*Size*	*Meaning*
tName	4	Type name, such as HELP
fName	4	File name, such as LOOP
fSize	4	File length in bytes
cksum	4	Checksum
address	4	base address of file in nonvolatile memory (flag in ls bit)
execAddr	4	Execution address for case of an active program file
versDate	6	Version date in BCD, as YrMoDaHrMnSc
copyCnt	2	Copy counter diagnostic for program files

Here is the current CODES table entry that refers to the HELPLOOP file:

```
0508:450E20 4845 4C50 4C4F 4F50  HELP LOOP
0508:450E28 0000 2740 0510 3EC0  fSize = 0x2740 bytes.
0508:450E30 720F 8481 0000 0000  address = 0x720F8480.
0508:450E38 0302 1114 1321 0000  versDate = Feb 11, 2003 at 1413:21.
```

At the moment, the CODES table entry number for this file is (0x450E20 - 0x450000)/32, or 0x71 = 113. Although this could change, it is unlikely to happen often. (I think.)

Now, how does one access memory, now that we have the base address for the HELPLOOP file? Suppose we have the following SSDN:

```
1D22 0508 7200 0000
```

The `Offset` word can be used with a request for this device, and because of the `0x0020` flag in the first word, it is shifted left 8 bits before internally being added to the base address `0x72000000`. This means that one can access memory from `0x72000000`—`0x72FFFFFF` by using the appropriate `Offset` value. In this specific example, one could use `Offset` = `0x0F84` to access memory starting at `0x720F8400`. One would request this file in pieces, probably, since it is about 10K bytes in size. The pieces would naturally be in units of 256 bytes. One might use blocks of 2048 bytes or 4096 bytes, say, with the last block being no more than is sufficient to reach the end of the file.

Once the file is captured, it may be cached, perhaps in a file sharing server, for increased efficiency of future access by the application. What also should be stored is the `CODES` table entry number where the match was found, plus the version date. Armed with that saved information, when the application is called up, it can request the single 32-byte `CODES` table entry whose entry number was saved, check that it is indeed the one for the `HELPLOOP` file, and check for a match on the version date. If all that agrees, the previously captured file can be retrieved. If there is no agreement on the saved `CODES` table index, then a new search of the `CODES` table will have to be done. If the table index was ok, but the version date does not match, it means that the file contents have changed, so the memory accesses will need to be repeated to capture the latest version of the file.